

[INICIAR SESIÓN](#)[NUESTROS PLANES](#)[TODOS LOS CURSOS](#)[FORMACIONES](#)[CURSOS](#)[PARA EMPRESAS](#)[ARTÍCULOS DE TECNOLOGÍA > FRONT END](#)

# Creando una aplicación Java Web con Servlet



Alex Vieira

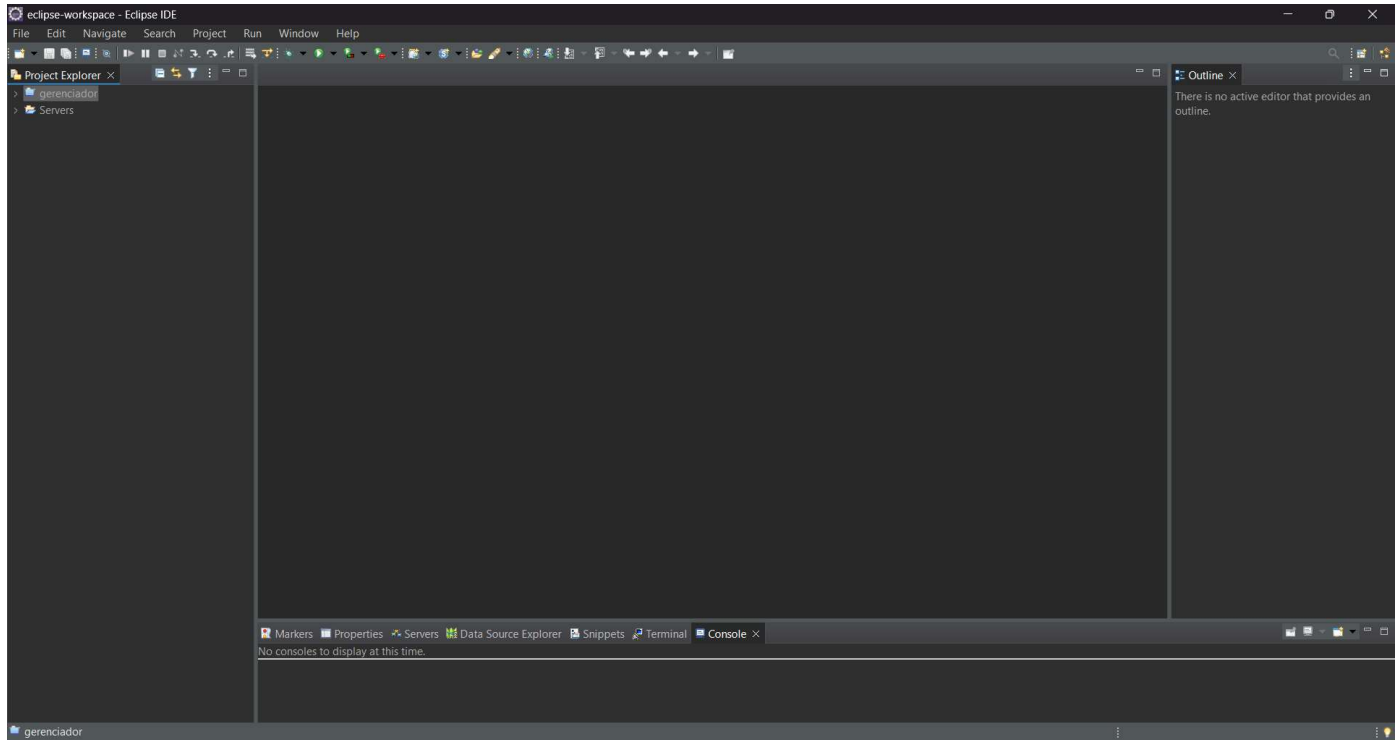
7 de Abril



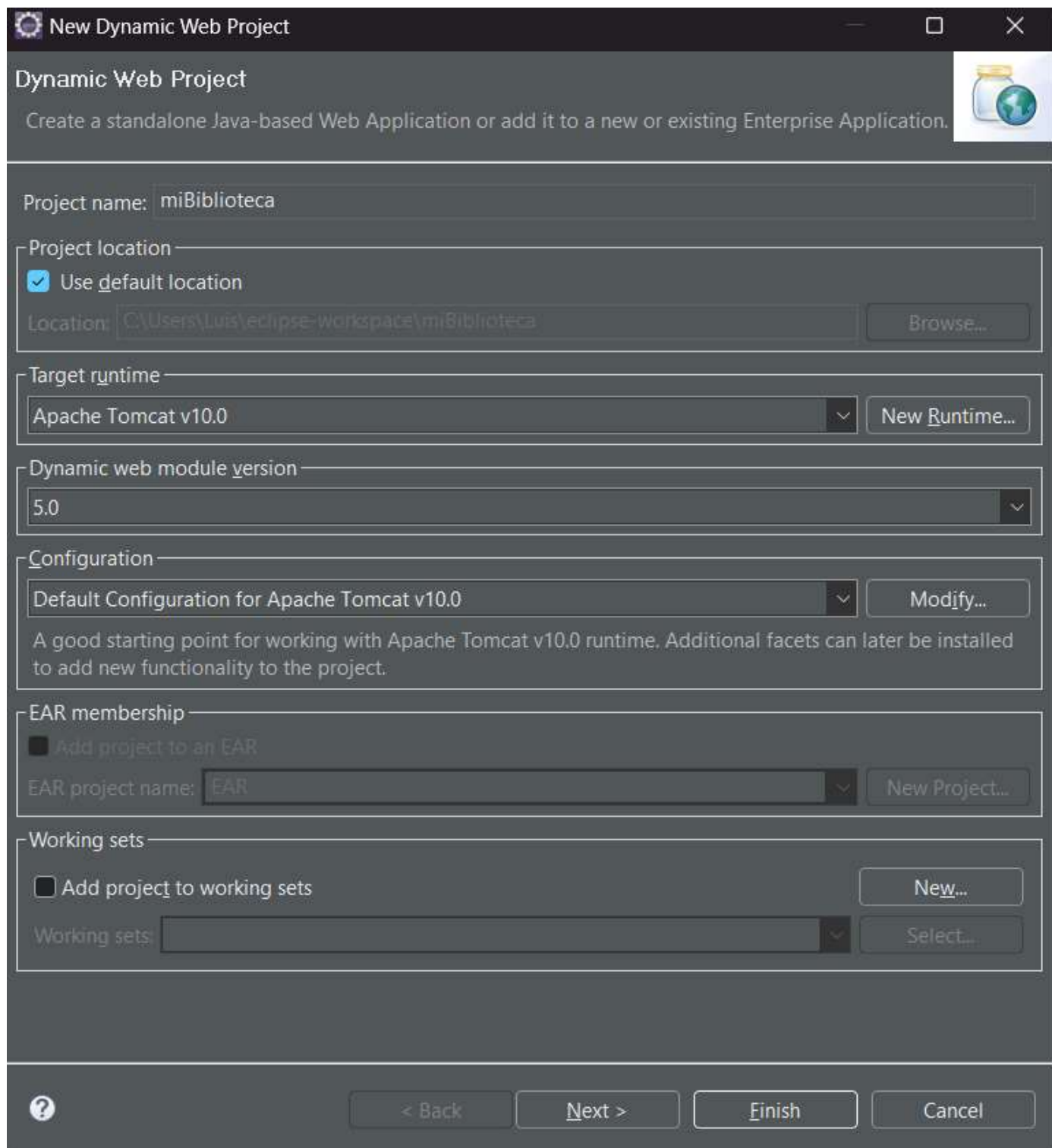
¡Voy a crear una aplicación web! O sea, una aplicación que se puede ejecutar en cualquier plataforma, ya sea en la computadora, un celular o cualquier dispositivo que funcione con cualquier navegador. Para ejecutar una aplicación web necesitamos un servidor de aplicaciones que se encargará de mantener nuestra aplicación en funcionamiento.

Actualmente existen varios servidores de aplicaciones y lenguajes de programación que podemos utilizar para nuestra aplicación, así que vamos a crear una aplicación Java Web y nuestro servidor de aplicaciones será Tomcat.

¡Para desarrollar esta aplicación usaremos el IDE Eclipse que nos facilitará la vida durante el desarrollo! Si no tiene Eclipse, puede descargar la versión de Java EE, descomprimir el archivo .zip y ejecutar Eclipse:



¡Necesitamos crear un proyecto Java Web! pero ¿cómo hacemos esto en nuestro Eclipse?, ¡Simple! Vaya al menú "File > New > Dynamic Web Project":



**New Dynamic Web Project**

Dynamic Web Project

Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v10.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

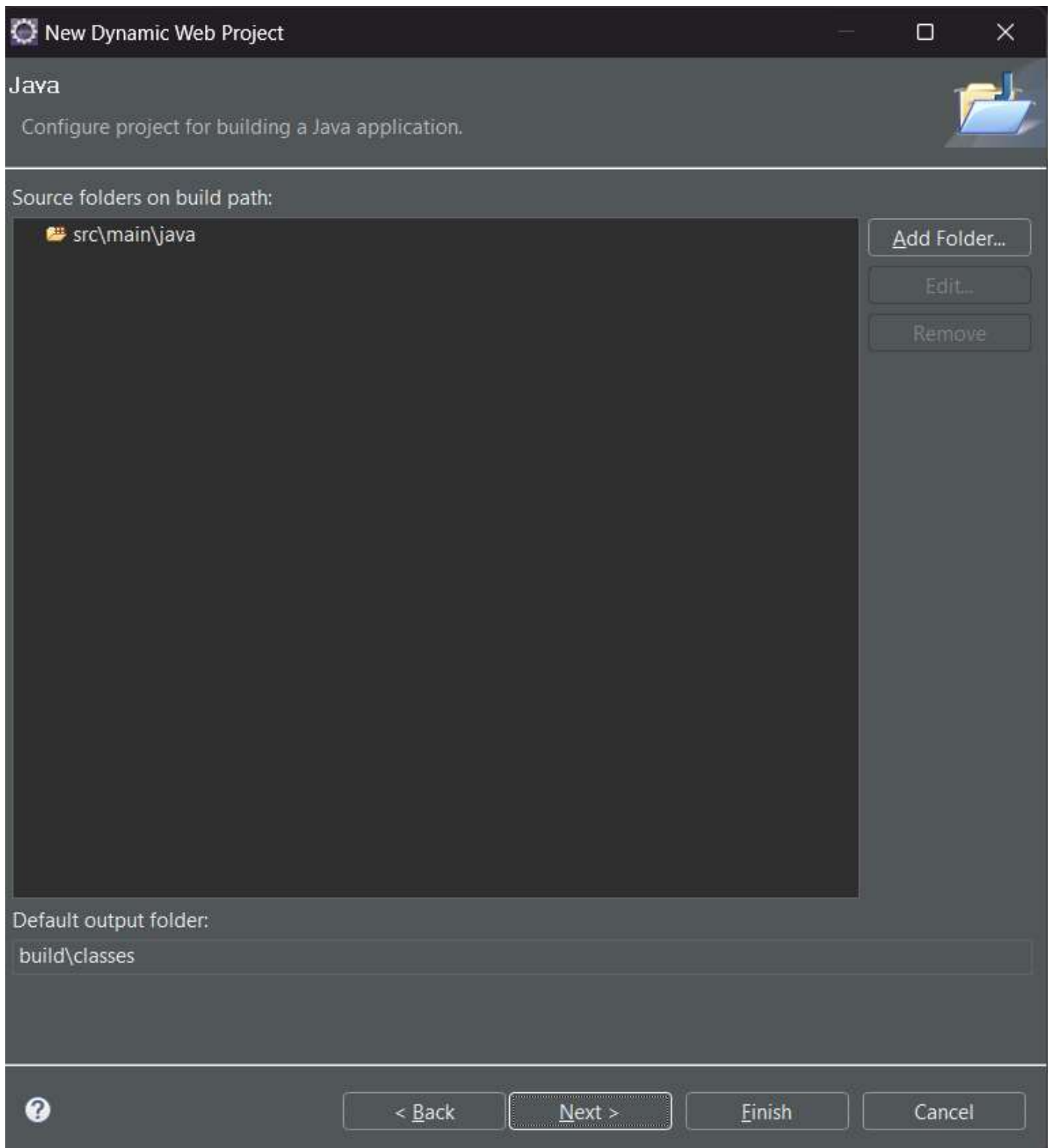
EAR project name:

Working sets

☐ Add project to working sets

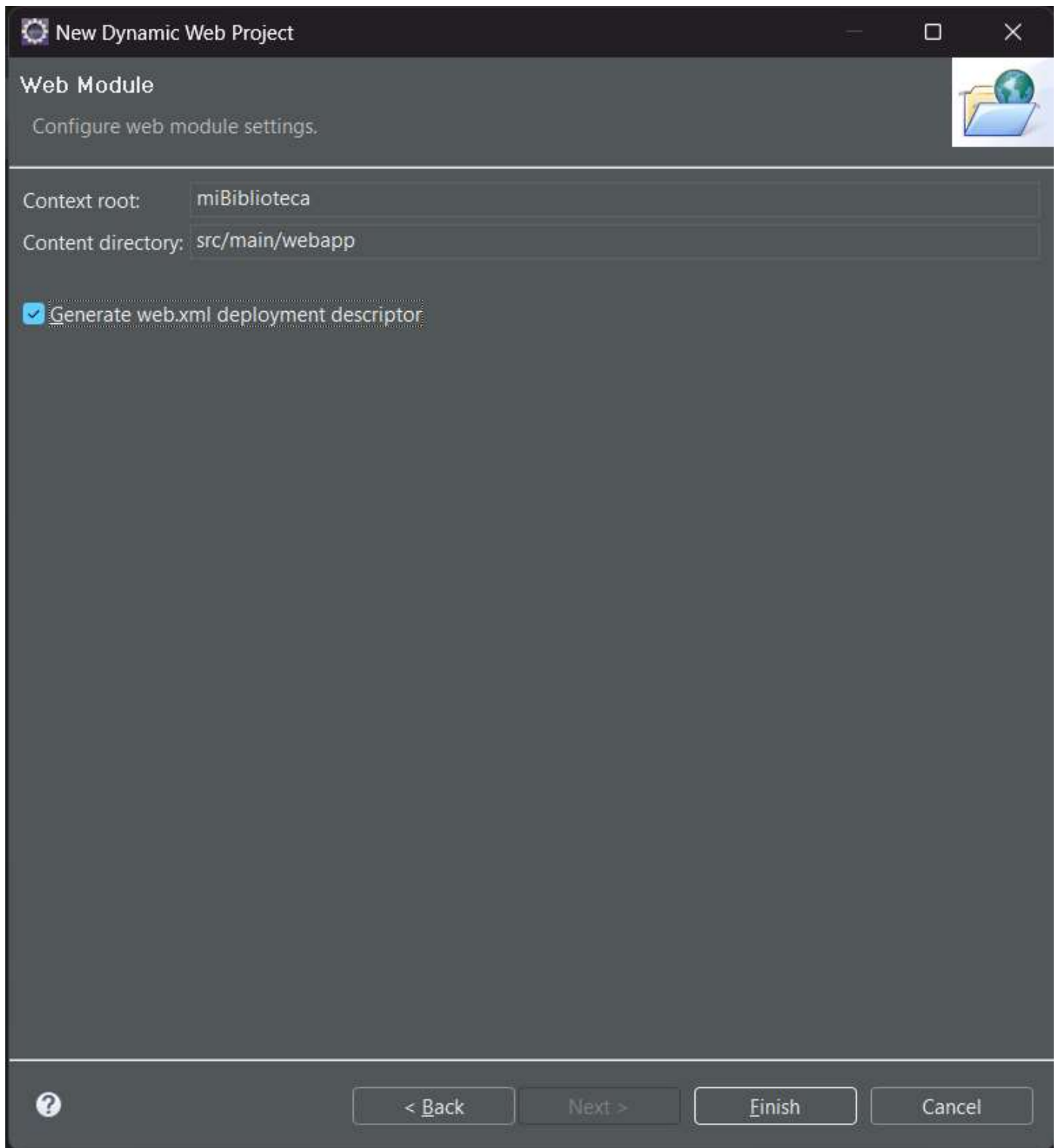
Working sets:

Le ponemos un nombre al proyecto, en este caso he puesto miBiblioteca, porque será una aplicación que registrará y listará todos mis libros. Después de colocar el nombre, haga clic en "Next":



Esta ventana nos sirve para la configuración de construcción del proyecto.

Dejaremos esta configuración estándar, luego haga click en “Next” nuevamente:




**New Dynamic Web Project**

**Web Module**  
Configure web module settings.

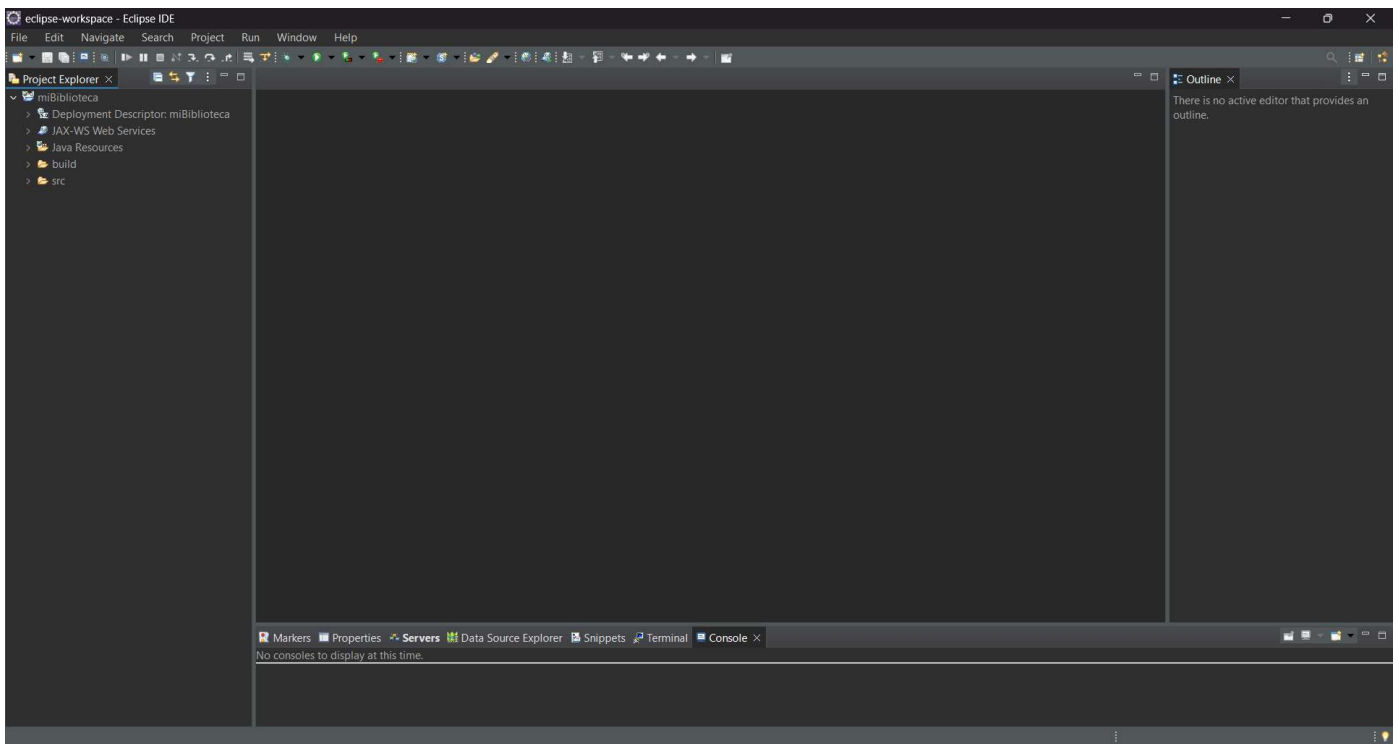
Context root:

Content directory:

☒ **Generate web.xml deployment descriptor**



Esta ventana nos sirve para las configuraciones del módulo, solo marcamos la opción "Generate web.xml deployment descriptor" para crear el archivo web.xml estándar que usaremos a continuación, finalmente damos clic en "Finish":



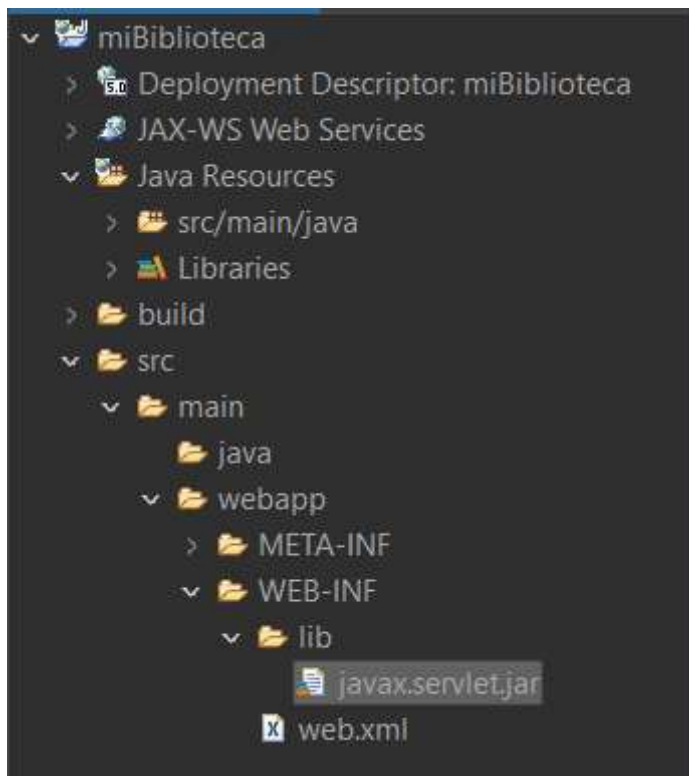
¡Observamos que nuestro proyecto fue creado! ¿Qué tenemos que hacer ahora? ¿Cómo podemos ejecutar nuestra aplicación?

Necesitamos, de alguna manera, configurar alguna clase para cumplir con las solicitudes de nuestra aplicación, entonces, ¿cómo podemos configurar esta clase?

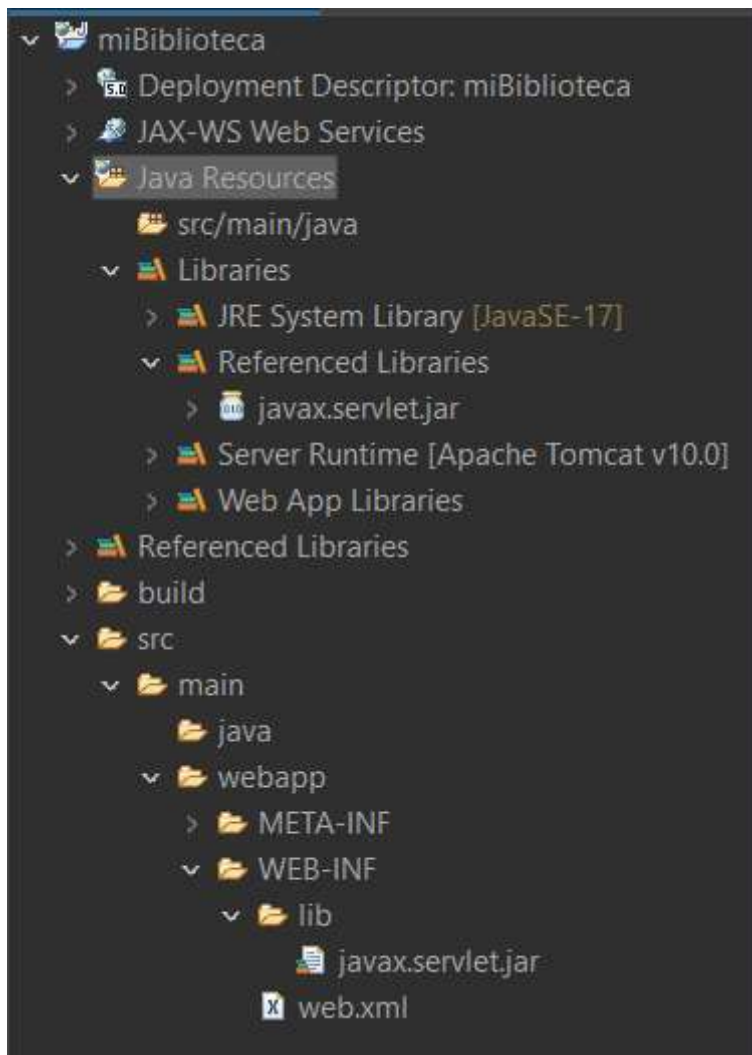
Hay varias formas de configurar esta clase y en este post usaremos la API de Servlet de Java para resolver esto.

Descargue la biblioteca Servlet 3.0 la opción con el nombre "[servlet-3\_0-final-jar\_and\_schema.zip]" y descomprima el archivo. Ahora necesitamos agregar Servlet a nuestro proyecto, que es solo el archivo .jar.

Vayamos a Eclipse y observemos que hay una carpeta llamada lib dentro de la carpeta "WebContent/WEB-INF/". Copie el archivo .jar y colóquelo en esta carpeta:



Haga clic en el botón derecho en el archivo .jar y elija la opción "Build Path > Add to Build Path". Ahora vayamos a la carpeta "Java Resource" y observemos que nuestro .jar se ha agregado a nuestra lista de bibliotecas en el proyecto.



Cuando agregamos una biblioteca dentro de una carpeta de nuestro proyecto, como fue en este caso de "WebContent/WEB-INF/lib/", tenemos la ventaja de exportar nuestro proyecto junto con las dependencias para su funcionamiento, o sea, todas las bibliotecas necesarias para que nuestra aplicación funcione en cualquier lugar.

¡Ahora podemos configurar nuestro primer Servlet! Entonces, creemos una nueva clase que representará nuestro Servlet, llamémoslo MainServlet:

```
1 package miBiblioteca;  
2  
3 public class MainServlet {  
4  
5 }  
6
```



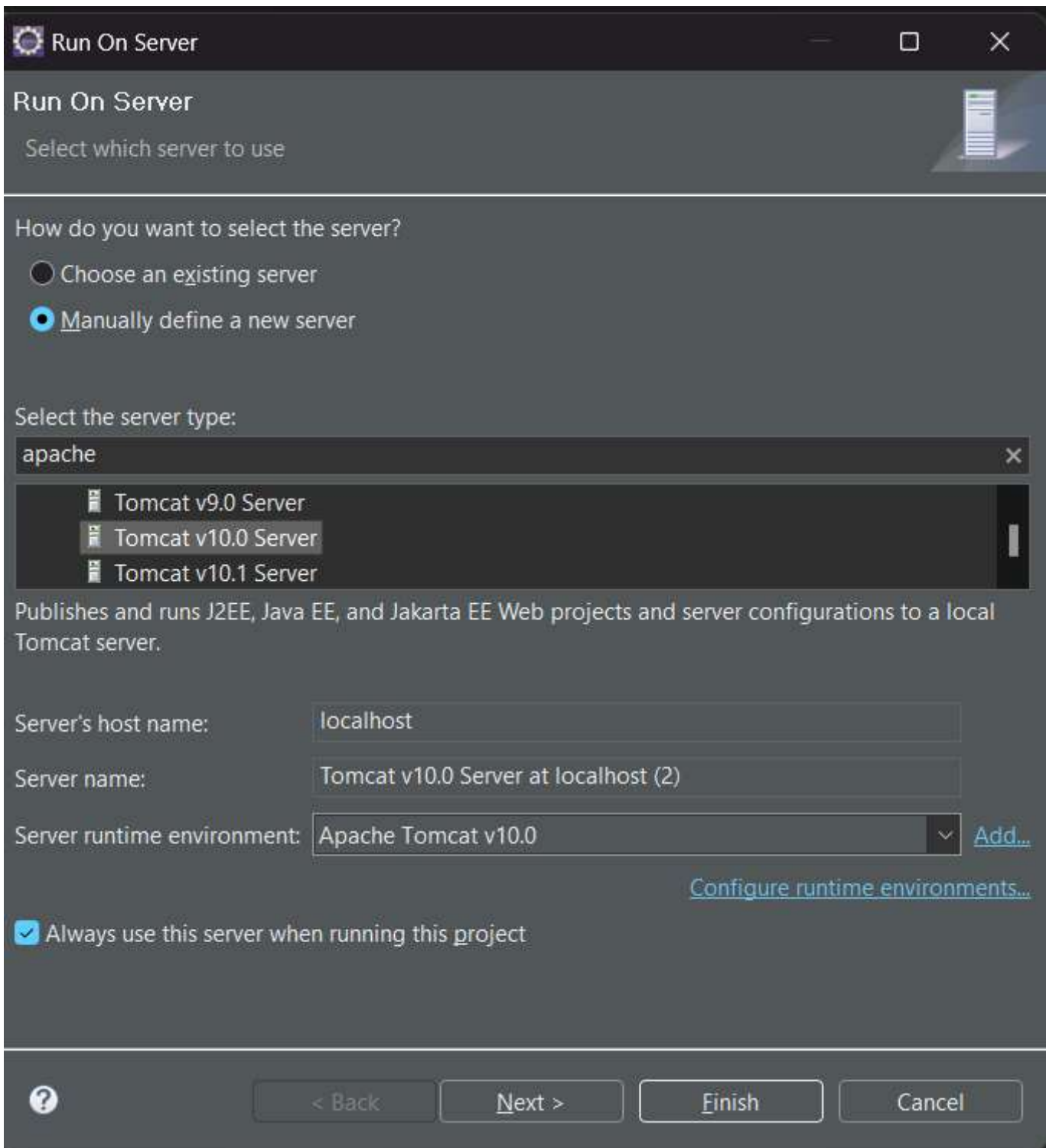
Para transformar una clase en un Servlet, necesitamos extender la clase HttpServlet:

```
1 package miBiblioteca;  
2  
3 import javax.servlet.http.HttpServlet;  
4  
5 public class MainServlet extends HttpServlet {  
6  
7 }  
8
```

Ahora necesitamos implementar el método que se encargará de recibir una solicitud a través de HTTP, que es doGET():

```
1 package miBiblioteca;  
2  
3 import java.io.IOException;  
4  
5 import javax.servlet.http.HttpServlet;  
6 import javax.servlet.http.HttpServletResponse;  
7  
8 import javax.servlet.ServletException;  
9 import javax.servlet.http.HttpServletRequest;  
10  
11 public class MainServlet extends HttpServlet {  
12  
13     @Override  
14     protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
15         throws ServletException, IOException { }  
16  
17 }  
18
```

¡Ya creamos nuestro primer Servlet! ¿Ejecutamos nuestra aplicación? Haga clic en el botón superior del proyecto "Run as > 1 Run On Server"



The image shows a 'Run On Server' dialog box with a dark theme. At the top, it says 'Run On Server' and 'Select which server to use'. Below this, it asks 'How do you want to select the server?' with two radio buttons: 'Choose an existing server' and 'Manually define a new server'. The second option is selected. Then, it says 'Select the server type:' and shows a list box with 'apache' entered. The list contains 'Tomcat v9.0 Server', 'Tomcat v10.0 Server' (which is highlighted), and 'Tomcat v10.1 Server'. Below the list, it says 'Publishes and runs J2EE, Java EE, and Jakarta EE Web projects and server configurations to a local Tomcat server.' Then, there are three input fields: 'Server's host name:' with 'localhost', 'Server name:' with 'Tomcat v10.0 Server at localhost (2)', and 'Server runtime environment:' with 'Apache Tomcat v10.0'. There is an 'Add...' link next to the last field and a 'Configure runtime environments...' link below it. At the bottom, there is a checkbox 'Always use this server when running this project' which is checked. At the very bottom, there are four buttons: '?', '< Back', 'Next >', 'Finish', and 'Cancel'.

Run On Server

Select which server to use

How do you want to select the server?

☐ Choose an existing server

☒ Manually define a new server

Select the server type:

apache

- Tomcat v9.0 Server
- Tomcat v10.0 Server
- Tomcat v10.1 Server

Publishes and runs J2EE, Java EE, and Jakarta EE Web projects and server configurations to a local Tomcat server.

Server's host name: localhost

Server name: Tomcat v10.0 Server at localhost (2)

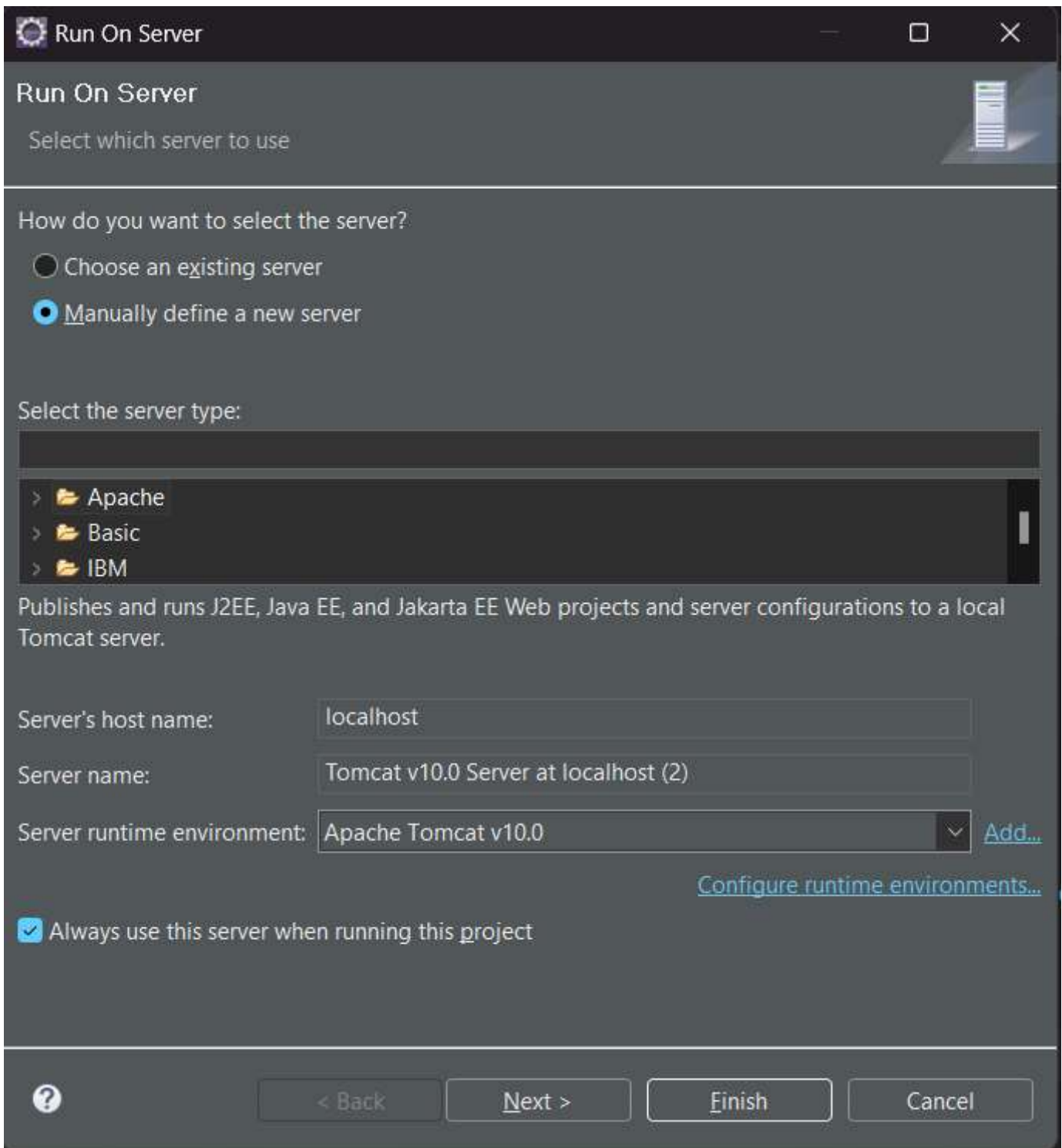
Server runtime environment: Apache Tomcat v10.0 [Add...](#)

[Configure runtime environments...](#)

☒ Always use this server when running this project

? < Back Next > Finish Cancel

¡Nos dimos cuenta de que ahora necesitamos configurar nuestro servidor de aplicaciones! Dentro del campo de "Select the server type:" Vemos la opción Tomcat, esta opción es la que vamos a usar y también elija en esta ventana siguiente haga clic en "Next">



The image shows a 'Run On Server' dialog box with a dark theme. At the top, it says 'Run On Server' and 'Select which server to use'. Below this, it asks 'How do you want to select the server?' with two radio buttons: 'Choose an existing server' (unselected) and 'Manually define a new server' (selected). Underneath, it says 'Select the server type:' followed by a list box containing 'Apache', 'Basic', and 'IBM'. Below the list box, it says 'Publishes and runs J2EE, Java EE, and Jakarta EE Web projects and server configurations to a local Tomcat server.' Then, there are three input fields: 'Server's host name:' with 'localhost', 'Server name:' with 'Tomcat v10.0 Server at localhost (2)', and 'Server runtime environment:' with 'Apache Tomcat v10.0'. To the right of the last field is an 'Add...' link. Below these fields is a link 'Configure runtime environments...'. At the bottom, there is a checkbox 'Always use this server when running this project' which is checked. At the very bottom, there are four buttons: a help button (question mark), '< Back', 'Next >', and 'Finish'. The 'Finish' button is highlighted.

Run On Server

Select which server to use

How do you want to select the server?

☐ Choose an existing server

☒ Manually define a new server

Select the server type:

- > Apache
- > Basic
- > IBM

Publishes and runs J2EE, Java EE, and Jakarta EE Web projects and server configurations to a local Tomcat server.

Server's host name: localhost

Server name: Tomcat v10.0 Server at localhost (2)

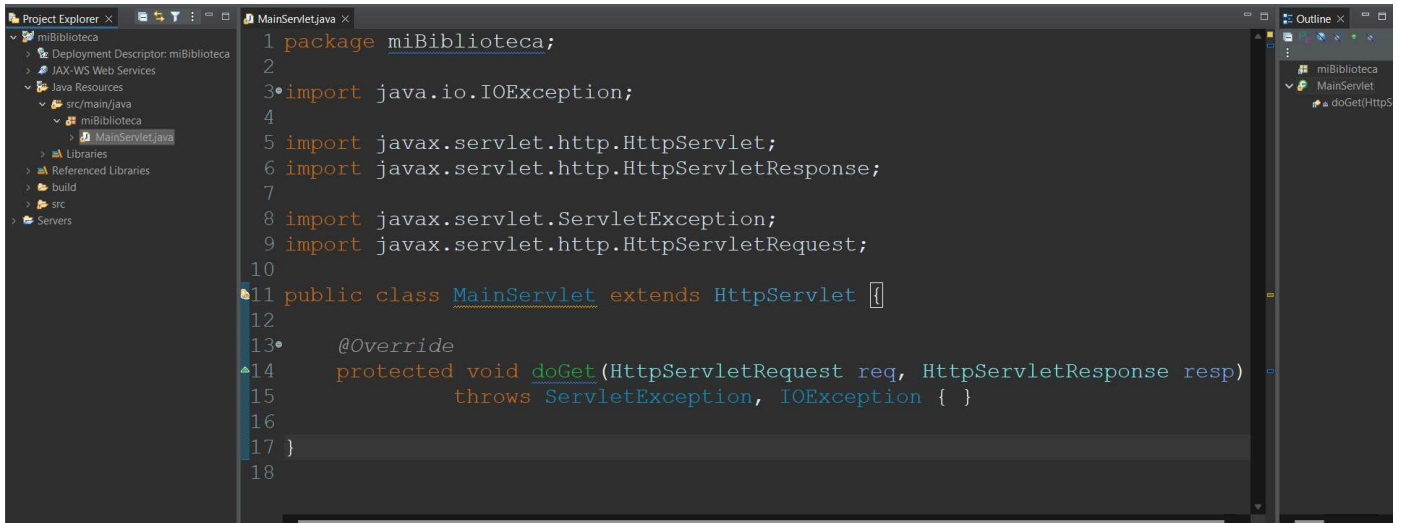
Server runtime environment: Apache Tomcat v10.0 [Add...](#)

[Configure runtime environments...](#)

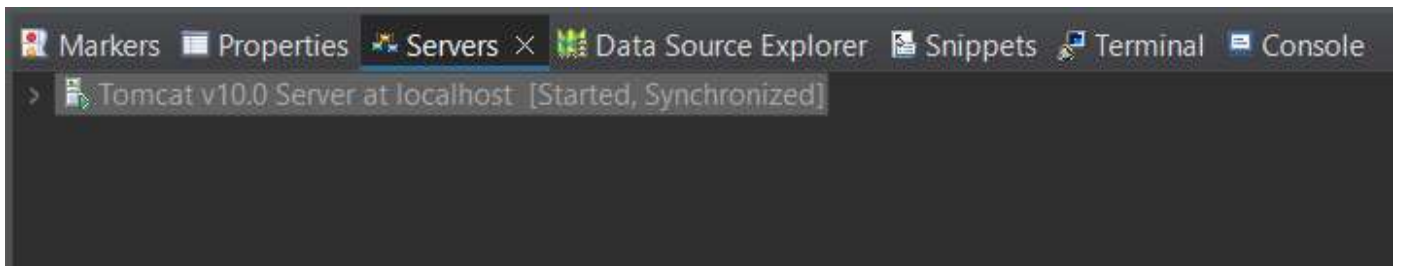
☒ Always use this server when running this project

[?](#) < Back Next > Finish Cancel

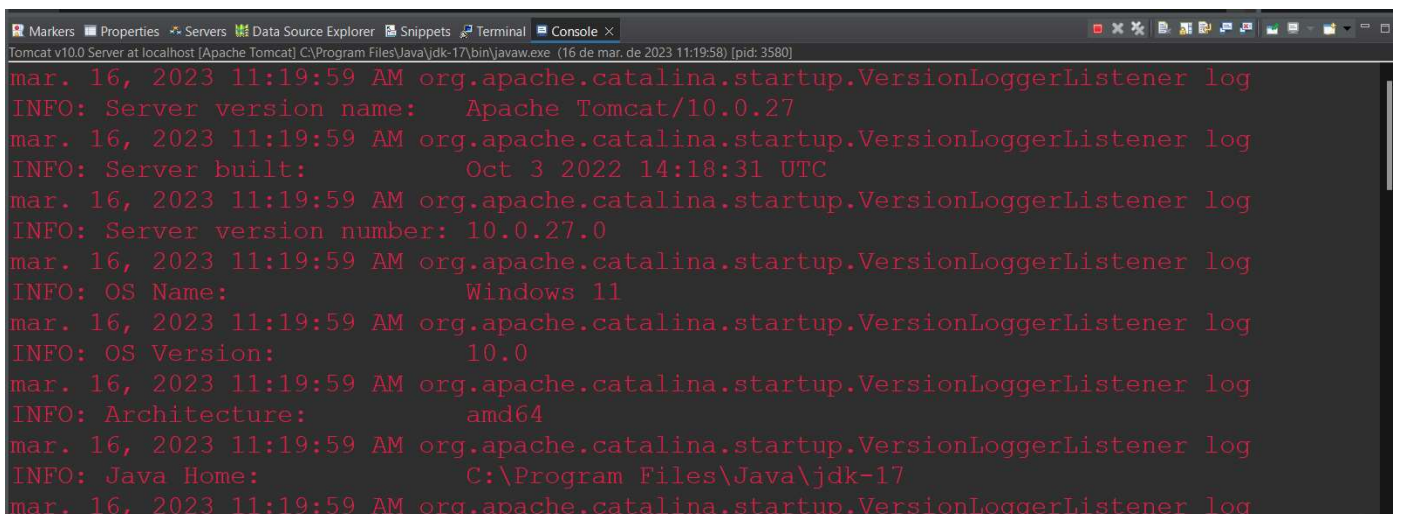
Luego hacemos click en "Finish"



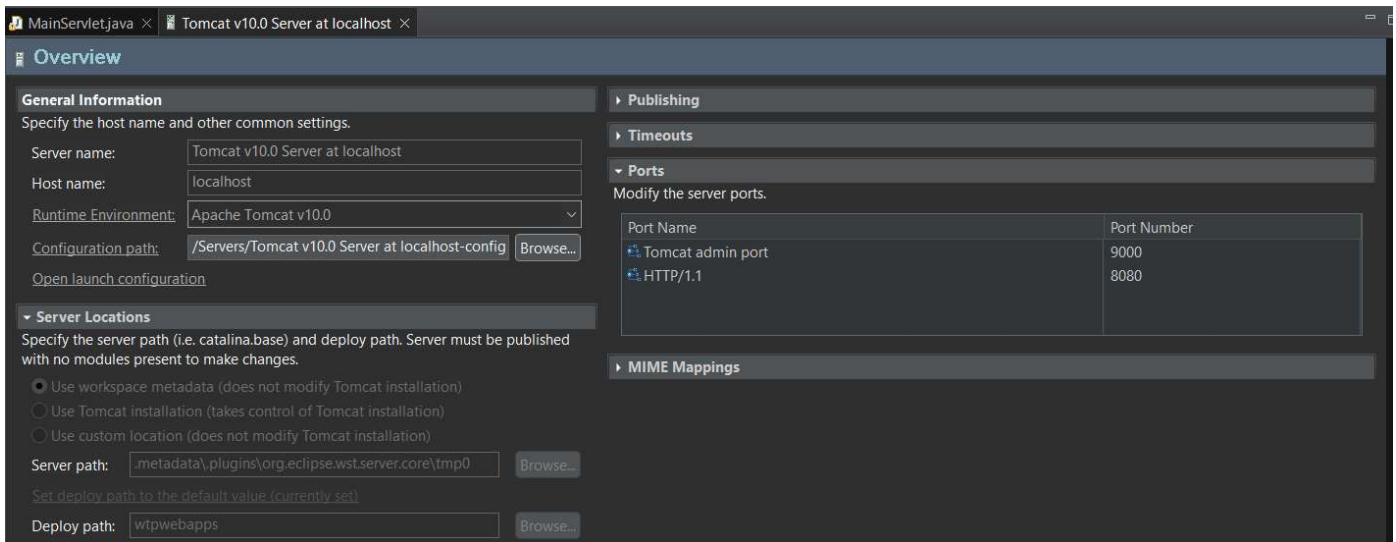
¡Ahora Tomcat está configurado! Puedes ver que se creó el proyecto Servers, que es precisamente el lugar donde Eclipse agregó nuestro Tomcat y también apareció una pestaña de Servers que indica que Tomcat está funcionando.



A demás de esto, verificamos en la pestaña console, veremos el log de Tomcat que muestra el momento en que se inició, si ocurrió algún problema o no:



Solo log de INFO, entonces, aparentemente, ¡todo está bien! Si hacemos clic sobre Tomcat en la view Servers, se abrirá el panel de configuración de nuestro servidor de aplicación:



Observamos que tenemos diversas opciones para configurar, por ello, las que nos interesa en este momento es precisamente el port name HTTP/1.1 que es justamente la puerta por la que Tomcat está tratando las solicitudes del protocolo HTTP, o sea, para acceder a nuestra aplicación a través de una URL del protocolo HTTP, ¡necesitará usar el puerto 8080!

¡Ahora podemos probar nuestra aplicación! Vamos al navegador, copiamos y obtenemos la siguiente URL:<http://localhost:8080/miBiblioteca> (En caso de que tengas otro nombre para tu proyecto, solo necesitas substituir la última parte de "miBiblioteca por el nombre que le pusiste y funcionaría") lo que significa que estamos accediendo a una URL vía HTTP en localhost (nuestra propia máquina) y en el puerto 8080. Finalmente, presiona Enter para hacer una llamada a la aplicación:



¿Error 404? ¡No encontré nada de nuestro proyecto! ¿Qué pasó? Además de crear un Servlet, necesitamos registrarlo en el archivo web.xml que es el archivo responsable de describir toda la información de implementación de deploy de nuestra aplicación. Así que registremos nuestro Servlet:



```

http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd (xsi:schemaLocation with c
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="https://
3
4 <• <servlet>
5 <servlet-name>Main Servlet</servlet-name>
6 <servlet-class>src.main.java.MainServlet</servlet-class>
7 </servlet>
8
9 <display-name>miBiblioteca</display-name>
10 <welcome-file-list>
11 <welcome-file>index.html</welcome-file>
12 <welcome-file>index.jsp</welcome-file>
13 <welcome-file>index.htm</welcome-file>
14 <welcome-file>default.html</welcome-file>
15 <welcome-file>default.jsp</welcome-file>
16 <welcome-file>default.htm</welcome-file>
17 </welcome-file-list>
18 </web-app>

```

Observamos que tuvimos que utilizar la etiqueta de nombre `servlet-name` para nombrar nuestro Servlet y una etiqueta `servlet-class` para especificar qué clase que representa este servlet que estamos registrando. También podemos ver que el nombre que necesitamos especificar es justamente el nombre completo de la clase, o sea, el nombre del package y después de la clase.

Si probamos nuevamente nuestra aplicación no responderá con la misma URL, ¡porque no hemos definido que responderemos a esa URL! Para configurar la dirección que nuestro Servlet atenderá, simplemente agregue la etiqueta `servlet-mapping` que informa el nombre de nuestro Servlet y la URL que se recibirá, en este caso una barra inclinada ("/"):

```

<servlet-mapping>
<servlet-name>Main Servlet</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>

```

Nuestro XML se quedaría así:

```

http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd (xsi:schemaLocation with c
1 <?xml version="1.0" encoding="UTF-8"?>
2 • <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="https://
3
4 • <servlet>
5   <servlet-name>Main Servlet</servlet-name>
6   <servlet-class>src.main.java.MainServlet</servlet-class>
7 </servlet>
8
9 • <servlet-mapping>
10  <servlet-name>Main Servlet</servlet-name>
11  <url-pattern>/[ ]/url-pattern|
12 </servlet-mapping>
13
14  <display-name>miBiblioteca</display-name>
15 • <welcome-file-list>
16   <welcome-file>index.html</welcome-file>
17   <welcome-file>index.jsp</welcome-file>
18   <welcome-file>index.htm</welcome-file>
19   <welcome-file>default.html</welcome-file>

```

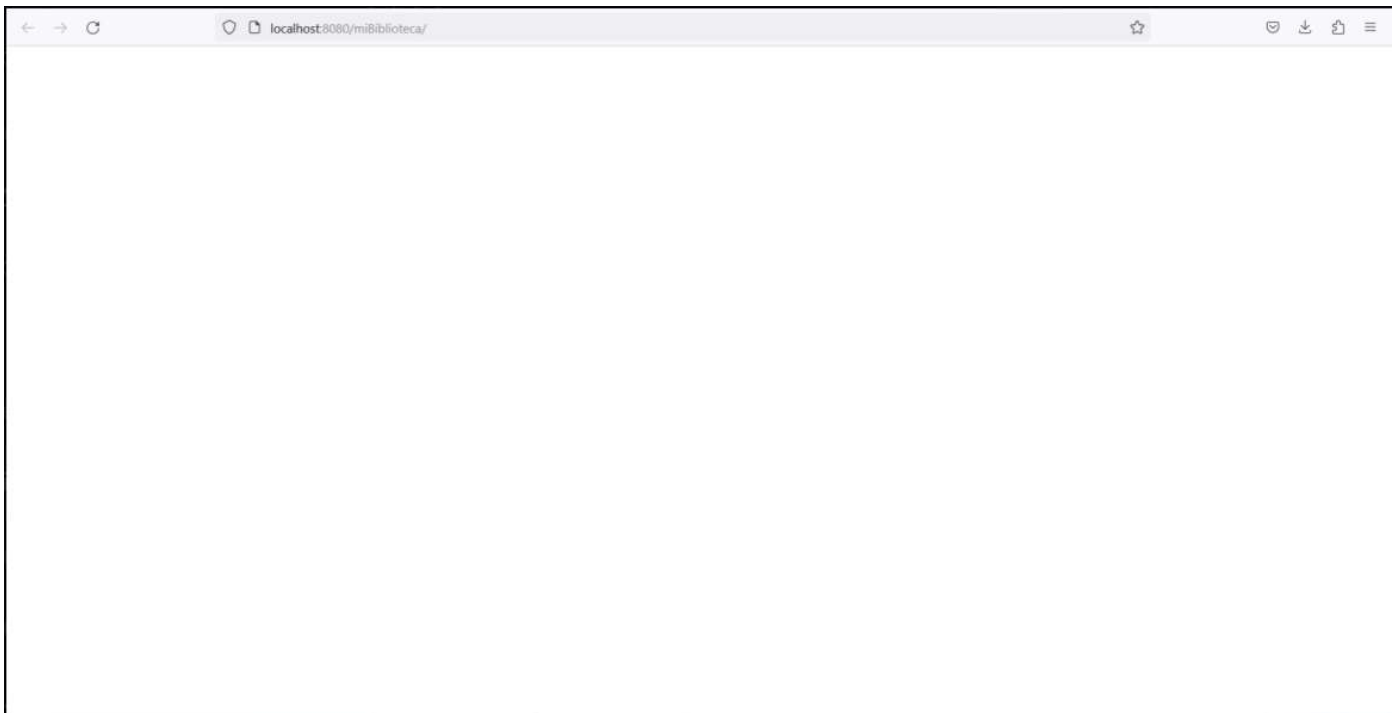
¿Vamos a probar nuestra aplicación de nuevo? Pero primero, agreguemos un mensaje para que se imprima al momento de ejecutar nuestro proyecto:

```

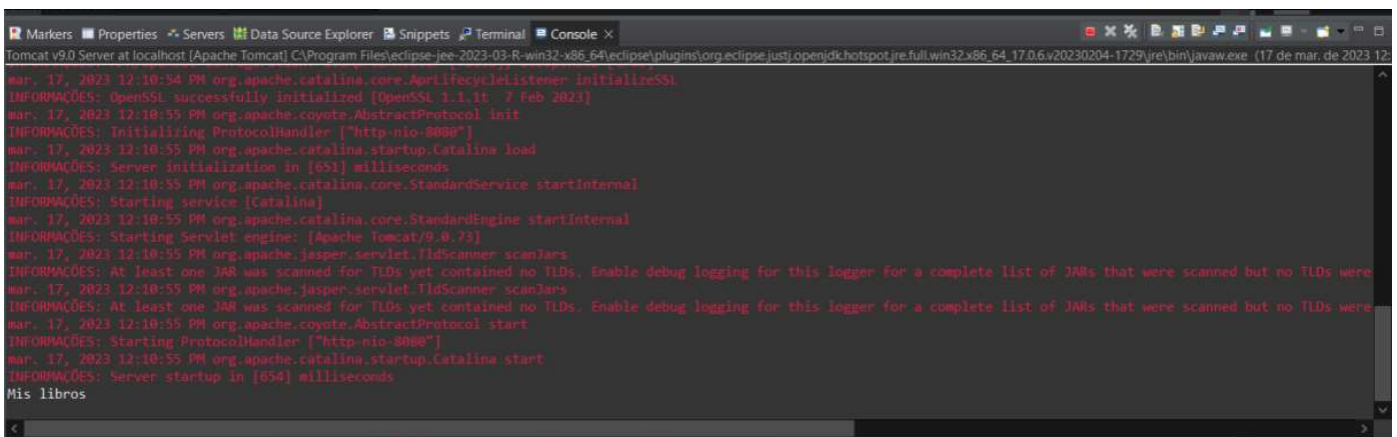
• @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    System.out.println("Mis Libros");
  }
}

```

Haga click en el botón derecho encima del proyecto "Run as > 1 Run on Server", ahora que Tomcat ya se inició, Eclipse va a preguntar si deseamos reiniciarlo, ponemos la opción de reiniciar para garantizar que todas las modificaciones que hicimos funcionen. Ahora, probamos nuestra URL: <http://localhost:8080/miBiblioteca>



Página en blanco? vamos a verificar nuestra consola:



Nuestro sysout fue imprimido! Pero lo ideal seria imprimir ese mensaje en el navegador! Como podemos hacer eso? Observamos que recibimos dos parametros en el método doGet(), HttpServletRequest req y HttpServletResponse resp:

- **HttpServletRequest** es el parámetro que representa las peticiones de nuestra aplicación, osea, todo lo que el cliente manda será recibido por medio de este parámetro.
- **HttpServletResponse** es el parámetro que representa las respuestas de nuestra aplicación, osea, todo lo que mandamos para el cliente será enviado por medio de este parámetro.

Esto significa que podemos usar HttpServletResponse para mandar un mensaje! y Haremos eso utilizando el método `getWriter()` que devuelve un objeto del tipo



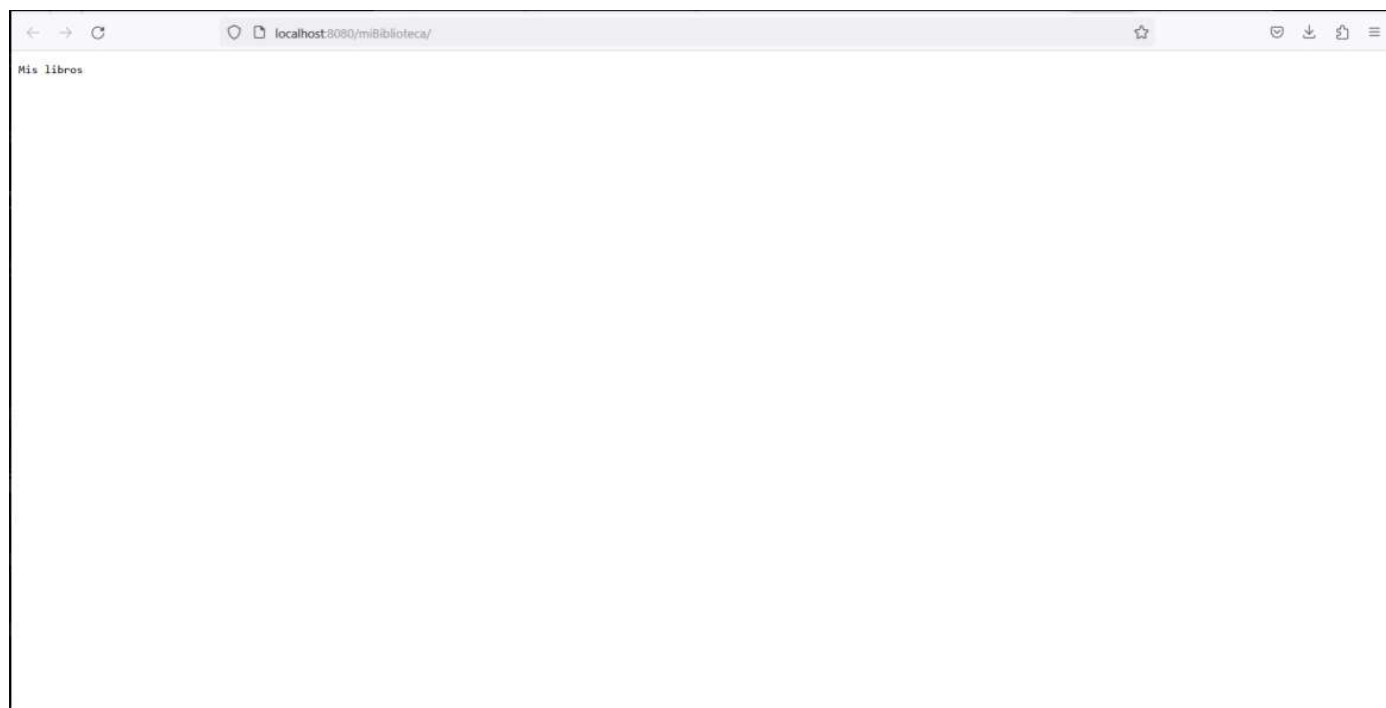
PrintWriter :

```
1 package miBiblioteca;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletResponse;
8
9 import javax.servlet.ServletException;
10 import javax.servlet.http.HttpServletRequest;
11
12 public class MainServlet extends HttpServlet {
13
14     @Override
15     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
16         throws ServletException, IOException {
17         PrintWriter writer = resp.getWriter();
18
19     }
20
21 }
```

Ahora falta solo imprimir utilizando el método print() :

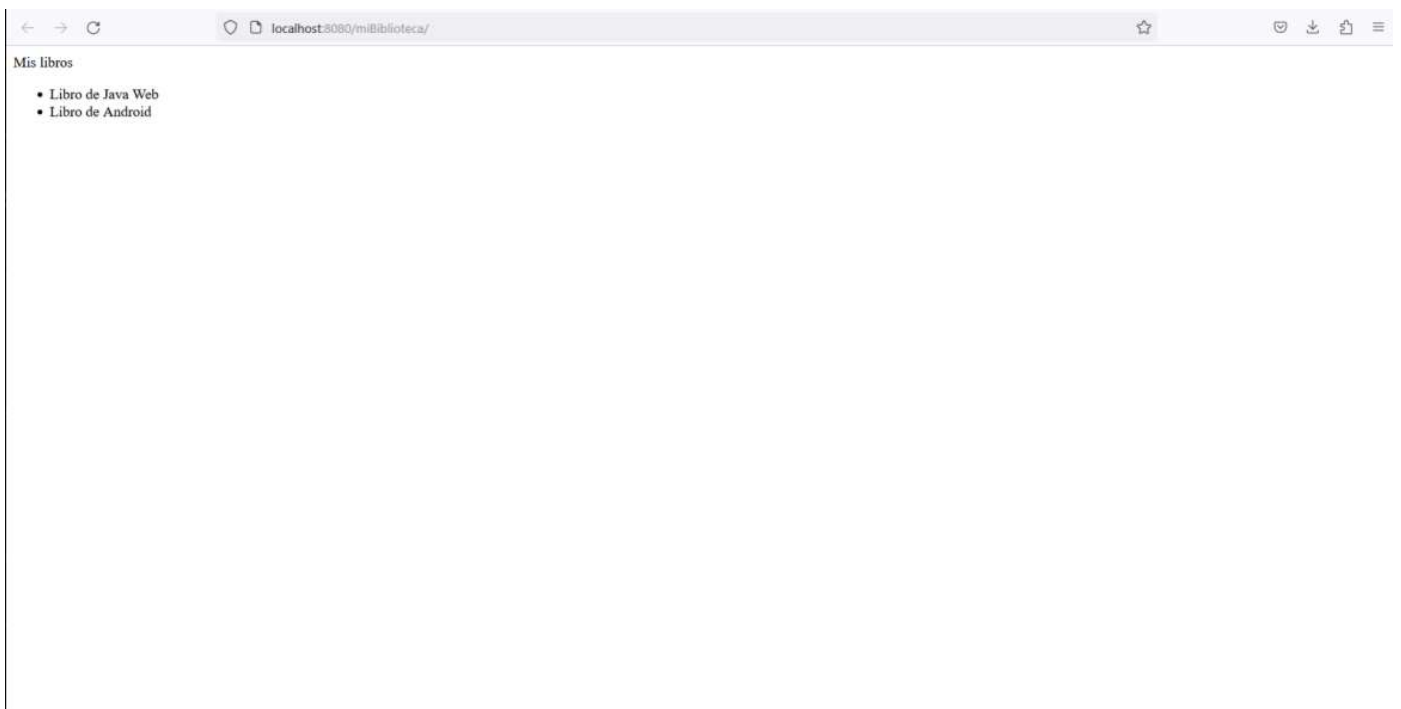
```
1 package miBiblioteca;
2
3 import java.io.IOException;
4
11
12 public class MainServlet extends HttpServlet {
13
14     @Override
15     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
16         throws ServletException, IOException {
17
18         resp.setContentType("text/html");
19
20         PrintWriter writer = resp.getWriter();
21         writer.print("Mis libros");
22
23     }
24
25 }
26
```

Vamos a comprobar nuevamente! Haga click con el botón derecho encima de Tomcat y elija la opción "restart" para reiniciarlo, entonces intentamos nuevamente el link:  
<http://localhost:8080/miBiblioteca>



Conseguimos enviar un mensaje! A demás de imprimir un texto, también podemos enviar tags en HTML, solo informando que la respuesta será en HTML por medio del método `setContentType()`, por ejemplo, para listar mis libros:

```
11
12 class MainServlet extends HttpServlet {
13
14     @Override
15     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
16         throws ServletException, IOException {
17
18         resp.setContentType("text/html");
19
20         PrintWriter writer = resp.getWriter();
21         writer.print("Mis libros");
22
23         writer.print("<ul>");
24
25         writer.print("<li>"); writer.print("Libro de Java Web"); writer.print("</li>");
26
27         writer.print("<li>"); writer.print("Libro de Android"); writer.print("</li>");
28
29         writer.print("</ul>");
30
31 }
```



¡Excelente! ¡Ya podemos imprimir los mensajes que queramos en nuestro navegador! ¡Es incluso mejor usando **HTML**!

En este post aprendimos cómo crear una aplicación Java Web básica usando **Eclipse** para **Java EE** y una **API** Servlet de Java. Vimos cómo podemos crear un proyecto para Java Web y cómo podemos agregar bibliotecas Servlet al proyecto por medio del **Build Path**.

Además, aprendimos que para crear un Servlet, necesitamos entender la clase `HttpServlet`, implementar el método `doGet()` y registrar nuestro servlet en el archivo `web.xml` que se encargará de describir toda la información de deploy de nuestra aplicación. Y por último,

vimos cómo enviar una respuesta en texto o en un **HTML** cada vez que alguien realiza una solicitud a nuestra aplicación.

¿Qué te pareció la API de Servlet? ¿Quieres aprender más sobre los servlets? En **Alura Latam** tenemos una [Formación Java](#), donde abordamos diferentes temas sobre Servlets y los principales fundamentos sobre programación de aplicaciones para Java Web!



### **Álex Felipe Victor Vieira**

Alex es instructor y desarrollador y tiene experiencia en Java, Kotlin, Android. Creador de más de 40 cursos, como Kotlin, Flutter, Android, persistencia de datos, comunicación con Web API, personalización de pantalla, pruebas automatizadas, arquitectura App y Firebase. Es experto en Programación Orientada a Objetos, siempre con el objetivo de compartir las mejores prácticas y tendencias en el mercado de desarrollo de software. Trabajó durante 2 años como editor de contenidos en el blog de Alura y hoy sigue escribiendo artículos técnicos.

Traducido para Alura Latam por **Luis Puig**

Cursos de Front End

ARTÍCULOS DE TECNOLOGÍA > FRONT END

**En Alura encontrarás variados cursos sobre Front End.  
¡Comienza ahora!**

**SEMESTRAL****US\$49,90**

un solo pago de US\$49,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 6 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**[Paga en moneda local en los siguientes países](#)

**ANUAL****US\$79,90**

un solo pago de US\$79,90

- ✓ 218 cursos
- ✓ Videos y actividades 100% en Español
- ✓ Certificado de participación
- ✓ Estudia las 24 horas, los 7 días de la semana
- ✓ Foro y comunidad exclusiva para resolver tus dudas
- ✓ Acceso a todo el contenido de la plataforma por 12 meses

**¡QUIERO EMPEZAR A ESTUDIAR!**[Paga en moneda local en los siguientes países](#)

Acceso a todos  
los cursos

Estudia las 24 horas,  
dónde y cuándo quieras

Nuevos cursos  
cada semana

## NAVEGACIÓN

PLANES

INSTRUCTORES

BLOG

POLÍTICA DE PRIVACIDAD

TÉRMINOS DE USO

SOBRE NOSOTROS

PREGUNTAS FRECUENTES

## ¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

## BLOG

PROGRAMACIÓN

FRONT END

DATA SCIENCE

INNOVACIÓN Y GESTIÓN

DEVOPS

AOVS Sistemas de Informática S.A  
CNPJ 05.555.382/0001-33

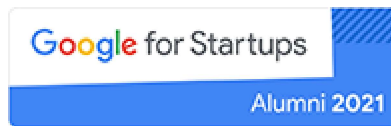
**SÍGUENOS EN NUESTRAS REDES SOCIALES**



## ALIADOS



En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.



Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

## CURSOS

### Cursos de Programación

Lógica de Programación | Java

### Cursos de Front End

HTML y CSS | JavaScript | React

### Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

### Cursos de DevOps

Docker | Linux

### Cursos de Innovación y Gestión



Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics |  
Liderazgo y Gestión de Equipos | Startups y Emprendimiento